In the claims:

Following is a complete set of claims as amended with this Response.

1. (Currently Amended) A method comprising:

reading a line of data from a file containing source code written in a high level hardware description language;

generating a stream of tokens from the ~~said~~ line of data, the ~~said~~ stream of tokens representing scan related macros ~~any of a specific type of macro~~ in the line of data as being expanded while other types of macros are not expanded;

generating a token object for each token, the token object including a visibility variable to represent whether a parser and an output module may view the respective token;

parsing the stream of tokens using a parser and with reference to respective token objects so that scan related macros are visible to the parser and other types of macros are not;

inserting commands representing operations to be performed by a macro into the ~~said~~ stream of tokens if a macro is visible ~~present~~; and

writing the stream of tokens to an output file using an output module and with reference to respective token objects so that scan related macros are expanded and other types of macros are not.

2. (Currently Amended) The method of claim 1, wherein generating a stream of tokens further comprises:

determining whether tokens are present in either an input file, a lookahead buffer, or a macro expansion list; and

responsive to finding tokens, reading the tokens first from the ~~said~~ lookahead buffer, then from the ~~said~~ macro expansion list, then from the ~~said~~ input file;

presenting the tokens to a parser so that any macro in the line of data appears to have been expanded.

3. (Previously Presented) The method of claim 1, wherein parsing further comprises:

reading a token;

determining a type of the read token;

responsive to determining that the read token is an end-of-line, processing an input line of tokens;

responsive to determining that the read token is not a symbol, adding the read token to a current line token list;

responsive to determining that the read token is a symbol that indicates a beginning of a macro definition, recording a macro name and the macro definition and adding the read token to a lookahead buffer; and

responsive to determining that the read token is a symbol that does not indicate a beginning of a macro definition, adding the read token to a current line token list.

4. (Cancelled)

5. (Previously Presented) The method of claim 1, wherein source code written in a high level language comprises a hardware description language (HDL) for representing hardware designs.

6. (Currently Amended) The method of claim 1, wherein the specific type of macro comprises a scan macro.

7. (Currently Amended) A method of scan insertion comprising:

reading a hardware description language (HDL) representation of a hardware design, the HDL including a plurality of macro definitions some of which relate to scan insertion;

creating a token stream based on the HDL representation that includes multifaceted tokens that can be hidden from or made visible to a subsequent parsing process by expanding the plurality of macro definitions and marking ~~making~~ tokens associated with scan macros as visible to the subsequent parsing process and marking other tokens as hidden, the multifaceted tokens being associated with a token object for each token, the token object including a visibility variable to represent whether a parser and an output module may view the respective token and a scan variable to represent whether the respective token is related to scan;

performing scan insertion using a parser by parsing those of the multifaceted tokens that are visible to the parser based on the ~~said~~ visibility variable and adding appropriate scan commands; and

using an output module to generate a scan inserted HDL file containing expanded versions of the macro definitions which are visible to the output module based on the visibility variable and which relate to scan insertion but that omits expanded versions of those that do not relate to scan insertion based on the scan variable.

8. (Previously Presented) The method of claim 7, wherein HDL comprises a high-level language.

9. (Previously Presented) The method of claim 7, wherein hardware design represents an integrated circuit design.

10. (Currently Amended) A system comprising:

a storage device having stored therein one or more routines for selectively expanding macros within source code; and

a processor coupled to the storage device for executing the one or more routines for selectively expanding macros within source code which, when executing the routine:

reads a line of data from a file containing source code written in a high level hardware description language;

generates a stream of tokens from the line of data, the stream of tokens representing scan related macros ~~any of a specific type of macro~~ in the line of data as being expanded while other types of macros are not expanded;

generates a token object for each token, the token object including a visibility variable to represent whether a parser and an output module may view the respective token;

parses the stream of tokens using a parser and with reference to respective token objects so that scan related macros are visible to the parser and other types of macros are not;

inserts commands representing operations to be performed by a macro into the stream of tokens if a macro is visible ~~present~~; and

writes the stream of tokens to an output file using an output module and with reference to respective token objects so that scan related macros are expanded and other types of macros are not.

11. (Previously Presented) The system of claim 10, wherein generating a stream of tokens further comprises:

determining whether tokens are present in either an input file, a lookahead buffer, or a macro expansion list; and

responsive to finding tokens, reading the tokens first from the lookahead buffer, then from the macro expansion list, then from the input file;

presenting the tokens to a parser so that any macro in the line of data appears to have been expanded.

12. (Previously Presented) The system of claim 10, wherein parsing further comprises:

reading a token;

determining a type of the read token;

responsive to determining that the read token is an end-of-line, processing an input line of tokens;

responsive to determining that the read token is not a symbol, adding the read token to a current line token list;

responsive to determining that the read token is a symbol that indicates a beginning of a macro definition, recording a macro name and the macro definition and adding the read token to a lookahead buffer; and

responsive to determining that the read token is a symbol that does not indicate a beginning of a macro definition, adding the read token to a current line token list.

13. (Canceled)

14. (Previously Presented) The system of claim 10, wherein source code written in a high level language comprises a hardware description language (HDL) for representing hardware designs.

15. (Previously Presented) The system of claim 10, wherein specific type of macro comprises a scan macro.

16. (Currently Amended) A machine-readable medium having stored thereon data representing sequences of instructions, the sequences of instructions which, when executed by a processor, cause the processor to selectively expand macros by:

reading a line of data from a file containing source code written in a high level hardware description language;

generating a stream of tokens from the line of , the stream of tokens representing scan related macros ~~any of a specific type of~~ ~~macro~~ in the line of data as being expanded while other types of macros are not expanded;

generating a token object for each token, the token object including a visibility variable to represent whether a parser and an output module may view the respective token;

parsing the stream of tokens using a parser and with reference to respective token objects so that scan related macros are visible to the parser and other types of macros are not;

inserting commands representing operations to be performed by a macro into the stream of tokens if a macro is visible ~~present~~; and

writing the stream of tokens to an output file using an output module and with reference to respective token objects so that scan related macros are expanded and other types of macros are not.

17. (Previously Presented) The machine-readable medium of claim 16, wherein generating a stream of tokens further comprises:

determining whether tokens are present in either an input file, a lookahead buffer, or a macro expansion list; and

responsive to finding tokens, reading the tokens first from the lookahead buffer, then from the macro expansion list, then from the input file;

presenting the tokens to a parser so that any macro in the line of data appears to have been expanded.

18. (Previously Presented) The machine-readable medium of claim 16, wherein parsing further comprises:

reading a token;

determining a type of the read token;

responsive to determining that the read token is an end-of-line, processing an input line of tokens;

responsive to determining that the read token is not a symbol, adding the read token to a current line token list;

responsive to determining that the read token is a symbol that indicates a beginning of a macro definition, recording a macro name and macro definition and adding the read token to a lookahead buffer; and

responsive to determining that the read token is a symbol that does not indicate a beginning of a macro definition, adding the read token to a current line token list.

19. (Cancelled)

20. (Previously Presented) The machine-readable medium of claim 16, wherein source code written in a high level language comprises a hardware description language (HDL) for representing hardware designs.

21. (Previously Presented) The machine-readable medium of claim 16, wherein specific type of macro comprises a scan macro.

22. (Previously Presented) A machine-readable medium having stored thereon data representing sequences of instructions, the sequences of instructions which, when executed by a processor, cause the processor to perform scan insertion by:

reading a hardware description language (HDL) representation of a hardware design, the HDL including a plurality of macro definitions some of which relate to scan insertion;

creating a token stream based on the HDL representation that includes multifaceted tokens that can be hidden from or made visible to a subsequent parsing process by expanding the plurality of macro definitions and making tokens associated with scan macros visible to the subsequent parsing process and marking other tokens as hidden, the multifaceted tokens being associated with a token object for each token, the token object including a visibility variable to represent whether a parser and an output module may view the respective token and a scan variable to represent whether the respective token is related to scan;

performing scan insertion using a parser by parsing those of the multifaceted tokens that are visible to the parser based on the visibility variable and adding appropriate scan commands; and

using an output module to generate a scan inserted HDL file containing expanded versions of the macro definitions which are visible to the output module based on the visibility variable and which relate to scan insertion but that omits expanded versions of those that do not relate to scan insertion based on the scan variable.

23. (Previously Presented) The machine-readable medium of claim 22, wherein HDL comprises a high-level language.

24. (Previously Presented) The machine-readable medium of claim 22, wherein hardware design represents an integrated circuit design.

25. (Previously Presented) The method of claim 1, wherein writing comprises:

writing expanded macro tokens to the output file if the macro is of the specific type of macro; and

writing an original macro call to the output file if the macro is not the specific type of macro.

26. (Previously Presented) The machine-readable medium of claim 16, wherein writing comprises:

writing expanded macro tokens to the output file if the macro is of the specific type of macro; and

writing an original macro call to the output file if the macro is not the specific type of macro.

27-32. (Canceled)

33. (Previously Presented) A method comprising:

reading lines of a hardware description language (HDL) representation of a hardware design, the HDL lines including a plurality of macro definitions some of which relate to scan insertion;

creating a token stream based on the HDL representation, each token including an IsScan variable to indicate whether the respective token is related to scan insertion;

performing scan insertion by parsing those of the tokens that are related to scan and adding scan commands; and

generating a scan inserted HDL file containing expanded versions of the macro definitions which relate to scan insertion but omitting expanded versions of the macro definitions that do not relate to scan insertion.

34. (Previously Presented) The method of claim 33, wherein creating a token stream comprises creating tokens that include a hidden token type variable to define whether the token is to be hidden from or made visible to a subsequent parsing process.

35. (Previously Presented) The method of claim 33, wherein the HDL is a high-level language.

36. (Previously Presented) The method of claim 33, wherein the hardware design represents an integrated circuit design.

37. (Previously Presented) The method of claim 33, wherein generating comprises:

receiving a token from a scan insertion module;

determining whether the token involves scan related changes; and

writing the token to an output file if the token is not scan related.

38. (Previously Presented) The method of claim 37 ~~33~~, further comprising determining whether more tokens are to be received from the scan insertion module and repeating determining whether the token involves scan related changes and writing the token to an output file until no tokens remain .

39. (Previously Presented) The method of claim 33, wherein creating a token stream comprises:

reading an HDL line from the HDL representation;

storing the lines in a lookahead list;

expanding macros that are related to scan and storing the expanded macros in a macro expansion list;

reading tokens first from the lookahead list, then from the macro expansion list, and then from the HDL file; and

passing the read tokens for parsing as if all the macros have been expanded.

40. (Previously Presented) The method of claim 33, wherein creating a token stream comprises expanding the included macros that are related to scan and storing the expanded macros in a macro expansion list before passing the tokens for parsing as if all the macros have been expanded.

41. (Previously Presented) The method of claim 33, wherein generating comprises writing the scan inserted tokens into the HDL file from a buffer that preserves the text of the original file.

42. (Currently Amended) The method of claim 33, further comprising parsing the tokens by:

determining whether a token is a macro name; and

expanding the token and [added] <u>adding</u> it to an expanded macro list if the token is a macro name[;]<u>.</u>

43. (Previously Presented) The method of claim 42 wherein parsing further comprises:

processing the token if the token is an end-of-line character;

adding the token to a current line token list if the token is found to be anything other than a symbol;

recording the macro name and definition in a lookahead buffer if the token is a symbol that begins a macro definition; and

adding the token to a current line token list if the token is a symbol does not begin a macro definition.

44. (Previously Presented) A machine-readable medium having stored thereon data representing sequences of instructions that, when executed by a machine, cause the machine to perform operations comprising:

reading lines of a hardware description language (HDL) representation of a hardware design, the HDL lines including a plurality of macro definitions some of which relate to scan insertion;

creating a token stream based on the HDL representation, each token including an IsScan variable to indicate whether the respective token is related to scan insertion;

performing scan insertion by parsing those of the tokens that are related to scan and adding scan commands; and

generating a scan inserted HDL file containing expanded versions of the macro definitions which relate to scan insertion but omitting expanded versions of the macro definitions that do not relate to scan insertion.

45. (Previously Presented) The machine-readable medium of claim 44, wherein creating a token stream comprises creating tokens that include a hidden token type variable to define whether the token is to be hidden from or made visible to a subsequent parsing process.

46. (Previously Presented) The machine-readable medium of claim 44, wherein generating comprises:

receiving a token from a scan insertion module;

determining whether the token involves scan related changes; and

writing the token to an output file if the token is not scan related.

47. (Previously Presented) The machine-readable medium of claim 44, wherein creating a token stream comprises:

reading an HDL line from the HDL representation;

storing the lines in a lookahead list;

expanding macros that are related to scan and storing the expanded macros in a macro expansion list;

reading tokens first from the lookahead list, then from the macro expansion list, and then from the HDL file; and

passing the read tokens for parsing as if all the macros have been expanded.

48. (Previously Presented) The machine-readable medium of claim 44, wherein generating comprises writing the scan inserted tokens into the HDL file from a buffer that preserves the text of the original file.

49. (Previously Presented) A scan insertion tool for a semiconductor hardware description language comprising:

a tokenizer module to read lines of an (HDL) representation of a hardware design, the HDL lines including a plurality of macro definitions some of which relate to scan insertion and to create a token stream based on the HDL representation, each token including an IsScan variable to indicate whether the respective token is related to scan insertion;

a scan insertion module to perform scan insertion by parsing those of the tokens that are related to scan and adding scan commands; and a file output module to generate a scan inserted HDL file containing expanded versions of the macro definitions which relate to scan insertion but omitting expanded versions of the macro definitions that do not relate to scan insertion.

50. (Previously Presented) The tool of claim 49, further comprising a parsing module to determine whether a token is a macro name, and to expand the token and add it to an expanded macro list if the token is a macro name.

51. (Previously Presented) The tool of claim 50 wherein the parsing module is further to process the token if the token is an end-of-line character, to add the token to a current line token list if the token is found to be anything other than a symbol, to record the macro name and definition in a lookahead buffer if the token is a symbol that begins a macro definition, and to add the token to a current line token list if the token is a symbol that does not begin a macro definition.

52. (Previously Presented) The tool of claim 50, wherein the tokenizer module creates tokens that include a hidden token type variable to define whether the token is to be hidden from or made visible to the parsing module.

53. (Previously Presented) The tool of claim 49, wherein the tokenizer module is further to read an HDL line from the HDL representation, to store the lines in a lookahead list, to expand macros that are related to scan and store the expanded macros in a macro expansion list, to read tokens first from the lookahead list, then from the macro expansion list, and then from the HDL file, and to pass the read tokens to the parsing module as if all the macros have been expanded.

54. (Previously Presented) The tool of claim 49, wherein the file output module is further to receive a token from a scan insertion module, to determine whether

the token involves scan related changes, and to write the token to an output file if the token is not scan related.